01 | 快速回顾：一个 C 程序的完整生命周期

2021-12-06 于航

《深入C语言和程序运行原理》 课程介绍 >

讲述：于航

时长 14:23 大小 13.18M

你好，我是于航。

在深入了解 C 语言、做到“知其所以然”之前，我们需要先做到“知其然”，也就是知道 C 语言是如何使用的。所以这一讲，我会从语法细节和语言特性、微观和宏观相结合的角度，带你快速、直观地回顾 C 语言的一些基础内容。而通过本讲的学习，你也会更容易理解这门课接下来的内容。

首先，我会带你回顾一个 C 程序从源代码编写到编译，再到最后运行的完整过程。除此之外，我还会用一段相对复杂的示例代码，来带你快速回顾 C 语言中最常见的那些语法及用方式。最后，我们还会从语言本身的角度，来探讨 C 语言与其他编程语言在编程范式上的不同之处。

学习这门课，需要怎样的实践开发环境？

在开始回顾这些内容之前，我要先向你推荐一些学习这门课时会用到的开发工具。你可以利用这些工具，自行编译和运行课程中给出的示例代码，以加深你对课程内容的理解。

对于编译工具，这门课会穿插使用运行于 x86-64 平台的 GCC 11.2 或 Clang 13.0.0 版本编译器。市面上有很多成熟的 C 编译器可以选择，但不同的编译器可能存在着所支持平台（类 Unix、Windows）以及 C 标准（C89、C99、C11、C17）上的差异，因此在选择时需要特别注意这些问题。这门课里使用的 GCC 和 Clang 都支持 C 语言的最新标准 C17，并且都可以运行在类 Unix 与 Windows 系统上。

当然，如果你在本地环境中没有安装上面这些编译器，那么也可以直接使用云编译器，比如 Godbolt。相较于本地编译器，云编译器即开即用，而且可以随时灵活切换不同的编译器版本。

至于 IDE，那些常用的都可以，不过推荐你选择 Visual Studio Code，因为它较为轻量，且目前提供的插件能力也足够进行 C 语言开发。最后，这门课里出现的所有代码，我都会统一放在这个代码库中，你可以根据需要自行获取。

用一个程序快速回顾 C 核心语法

为了让你比较完整地回顾 C 语言的核心语法，我设计了一个相对复杂的 C 语言程序作为例子。在这里，你可以先试着阅读这段代码，思考下 C 语言的使用方式。代码如下所示：

复制代码

```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <assert.h>
#include <stdbool.h>

// 定义用到的宏常量与宏函数；
#define BOOL_TRUE 1
#define BOOL_FALSE 0
#define typename(x) _Generic((x), \
  unsigned short: "unsigned short int", \
  unsigned long: "unsigned long int", \
  default: "unknown")

// 定义枚举类型 IP_ADDR_TYPE，用于表示联合中生效的字段；
```

```c
typedef enum { Host, IP } IP_ADDR_TYPE;
// 定义结构 CONN；
typedef struct {
  size_t id;
  uint16_t port;
  bool closed;
  IP_ADDR_TYPE addr_type;
  union {
    char host_name[256];
    char ip[24];
  };
} CONN;

// 定义函数 findAddr，用于打印 CONN 对象的信息；
inline static const char* findAddr(const CONN* pip) {
  // 运行时断言，判断传入的 CONN 指针是否有效；
  assert(pip != NULL);
  return pip->addr_type == Host ? pip->host_name : pip->ip;
}
// 入口函数；
int main(int argc, char* argv[]) {
  // 静态断言，判断 CONN 对象的大小是否符合要求；
  static_assert(sizeof(CONN) <= 0x400, "the size of CONN object exceeds limit.
  // 构造一个数组，包含三个 CONN 对象；
  const CONN conns[] = {
    [2] = { 1, 80, BOOL_TRUE, IP, { .ip = "127.0.0.1" } },
    [0] = { 2, 8080, BOOL_FALSE, IP, { .ip = "192.168.1.1" } },
    { 3, 8088, BOOL_FALSE, Host, { .host_name = "http://localhost/" } }
  };
  // 遍历上述 CONN 数组，并打印其中的内容；
  for (size_t i = 0; i < (sizeof(conns) / sizeof(CONN)); ++i) {
    printf(
      "Port: %d\n"
      "Host/Addr: %s\n"
      "Internal type of `id` is: %s\n\n",
      conns[i].port,
      findAddr(&conns[i]),
      typename(conns[i].id)
    );
  }
  return EXIT_SUCCESS;
}
```

这段代码用到了横跨 K&R C 到 C17 标准的许多语言特性，创建了多个基于自定义类型构建的对象，并在程序的最后将这些对象的相关信息打印了出来。

下面，就来跟着我一起梳理这段代码中用到的 C 语法特性吧。我会按照程序代码的执行顺序，来分别介绍每一个执行步骤中涉及到的关键语言知识点。其中，相关的语言结构和语

法特性可以被分为下面这些类别。

入口函数

现在，让我们来仔细观察这个程序。首先，我们的目光来到第 31 行上名为 main 的函数。

所有的 C 程序都会使用 main 函数作为入口函数。入口函数，就是指程序开始运行时，代码中会被首先调用的那个函数。在 main 函数中，我们可以通过它接收到的实际参数，来选择性地访问程序在开始运行时，由用户传递给程序的外部参数。

main 函数在执行结束时会返回一个整数，用于表示程序执行完毕时的状态，通常返回数字 0 表示程序正常退出，返回其他数字则代表异常退出。为了保持代码的可读性，这里我们使用标准库中定义的宏常量 EXIT_SUCCESS ，作为程序退出的返回值。顾名思义，这个宏常量对应的实际值就是数字 0。

数组

接下来，我们来到第 33 行。可以看到，在 main 函数内部，我们使用了“括号列表（brace-enclosed lists）”的方式，完成了对数组 conns 的初始化过程。

而在初始化列表中，我们还使用了指派初始化（为初始化列表中的项设定“指派符”）的方式，来明确指定这些项在数组中的具体位置。比如这里第一项对应的 “[2]” ，就表示将该项设置为数组 conns 中的第 3 个元素（索引从 0 开始）。

数组定义完毕后，第 44 到第 46 行的代码访问了其内部存放的元素。这里我们直接使用方括号加索引值的语法形式做到了这一点。

结构与联合

数组 conns 内部，存放有若干个类型为 CONN 的结构对象。在 C 语言中，结构和联合（有时也被称为结构体与联合体）通常用来组织复杂类型的自定义数据。在结构中，所有定义字段的对应数据按照内存连续的方向排列；而在联合中，定义的字段同一时间只会有一个“生效”。

观察第 15 行到 24 行，可以看到：在我们对结构 CONN 的定义过程中，使用了来自 C99 标准的 `_Bool` 类型（这里的宏 bool 会展开为该类型），以及来自 C11 标准的匿名联合

体。

第 34 到第 36 行，在我们对结构 CONN 对象的初始化过程中，也同样使用了类似数组的括号列表初始化，以及指派初始化。但和前面数组初始化不同的是，这里的指派是针对结构与联合类型内部的成员字段的，因此需要使用 “.” 符号来引用某个具体成员，而非数组所使用的形式。

控制结构

在这段代码的第 39 行，我们使用了 for 语句以循环的形式遍历了数组 conns 中的内容。除此之外，C 语言中常用的控制结构还有 switch 语句、while 语句、以及 goto 语句等等。这些语句分别以选择、迭代，及跳转这三种不同方式控制着程序的实际执行逻辑。而程序本身也可以在这些控制语句的灵活组合下变得更加复杂。

指针

指针是 C 语言中最危险但也最强大的“武器”之一。借助指针，我们能够灵活地操控程序享有的内存资源。

在上面代码的第 45 行，我们将数组 conns 中各个元素的地址传递给了函数 findAddr，而该函数则接收一个指向 CONN 类型对象的常量指针。所以，通过该指针，我们无法在函数内部修改指针所指向对象的值。而这在一定程度上保证了函数仅能够拥有足够完成其任务的最小权限。

宏

编译器对 C 源代码的处理过程分为几个阶段，其中，宏是最先被处理的一个部分。在这段代码的开头处，我们通过宏指令 “#include” 引入了程序正常运行需要的一些外部依赖项，这些引入的内容会在程序编译时得到替换。随后，我们又通过 “#define” 指令定义了相应的宏常量与宏函数，而其中的宏函数 typename 则使用到了 C11 标准新引入的 `_Generic` 关键字，以用来实现基于宏的泛型。

断言

在这段代码的第 32 行，我们使用了 C11 标准中提供的静态断言能力，来保证结构类型 CONN 的大小不会超过一定的阈值。而在代码的第 27 行，我们还使用了运行时断言来保

证传递给函数 findAddr 的 CONN 对象指针不为空。

在 C 代码中，我们通常会使用断言，来对某种需要支持程序正常运行的假设性条件进行检查。而当条件不满足时，则在程序编译或运行时终止，并向用户抛出相应的错误信息。C 语言提供静态与动态两种类型的断言，其中静态断言会在代码编译时进行检查；而动态断言则会在程序运行过程中，执行到该断言语句时再进行检查。

函数内联

在函数 findAddr 的定义代码中，我们为其添加了名为 `inline` 的关键字。通过使用该关键字，我们可以“建议”编译器将该函数的内部逻辑直接替换到函数的调用位置处，以减少函数调用时产生的开销。这种方式通常使用在那些函数体较小，且会被多次调用的函数上，以产生较为显著的性能提升。

其他特性

除了上面提到的内容，这段代码中还涉及到了一些基本的 C 语言特性：

使用 `const` 定义只读变量；

使用 `typedef` 定义新类型；

使用 `static` 声明静态函数；

使用各类运算符；

调用标准库函数；

使用 `enum` 定义枚举类型；

……

这些语法特性本身比较常用，且概念较为简单，这里我就不再单独介绍了。如果你对其中的一些特性感到陌生，可以选择在 GeeksforGeeks网站上直接查找特定主题并学习，或者查阅《C Primer Plus》这些入门书籍。

到这里，我们就把 C 语言的核心语法大致捋了一遍。你可以看到，C 语言的语法并不复杂。C 语言在设计上就十分精简，截止到 C17 标准，语言本身也仅有 44 个关键字。C 语

言的强大并不是源于复杂的语法设计，相反，简单的语法给了 C 开发者更高的自由度，让我们可以更加灵活地设计程序的运行逻辑。

C 语言的编程范式是怎样的？

抛开语法细节，从总体上来看，C 语言是一种“命令式”编程语言，和它类似的还有 Java、C#、Go 等语言。

命令式编程（Imperative Programming）是这样一种编程范式：使用可以改变程序状态的代码语句，描述程序应该如何运行。这种方式更关注计算机完成任务所需要执行的具体步骤。

下面我们来看一个例子。对于“从一个包含有指定数字的集合中，筛选出大于 7 的所有数字”这个需求，按照命令式编程的思路，我们需要通过编程语言来告诉计算机具体的执行步骤。

以 C 语言为例，解决这个需求的步骤可能会是这样：

1. 使用数组，构造一块可以存放这些数字的内存空间；
2. 使用循环控制语句，依次检查内存中的这些数字是否满足要求（即大于 7）；
3. 对于满足要求的数字，将它们拷贝到新的内存空间中，暂存为结果。
   对应的代码可能如下所示：

复制代码

```
#define ARR_LEN 5
int main(void) {
  int arr[ARR_LEN] = { 1, 5, 10, 9, 0 };
  for (int i = 0; i < ARR_LEN; ++i) {
    if (arr[i] > 7) {
      // save this element somewhere else.
    }
  }
  return 0;
}
```

相对于命令式编程语言，其他语言一般会被归类为“声明式”编程语言。声明式编程（Declarative Programming）也是一种常见的编程范式。**不同的是，这种范式更倾向于**

表达计算的逻辑，而非解决问题时计算机需要执行的具体步骤。

比如说，还是刚才那个需求，在使用声明式编程语言时，对应的解决步骤可能是：

1. 构建一个容器来存放数据；
2. 按照条件对容器数据进行筛选，并将符合条件的数据作为结果返回。
   如果以 JavaScript 为例，对应的代码可能如下所示：

复制代码

```
let arr = [1, 5, 10, 9, 0]
let result = arr.filter(n => n > 7)
```

可以看到的是，相较于命令式编程，声明式编程更倾向于表达在解决问题时应该做什么（构建容器、筛选），而不是具体怎么做（分配内存、遍历、拷贝）。

通常来说，命令式编程语言和声明式编程语言的差异，主要体现在两者的语言特性相较于计算机指令集的抽象程度。其中，命令式编程语言的抽象程度更低，这意味着该类语言的语法结构可以直接由相应的机器指令来实现。而声明式编程语言的抽象程度更高，这类语言更倾向于以叙事的方式来描述程序逻辑，开发者无需关心语言背后在机器指令层面的实现细节。两种语言在使用上各有其适用场景，并无孰好孰坏之分。

那么，C 语言作为一种低抽象层次的命令式编程语言，它的各类语法结构是如何对应到不同机器指令的？我会在下一个模块“语法核心实现篇”里为你详细介绍。

C 程序的编译和运行

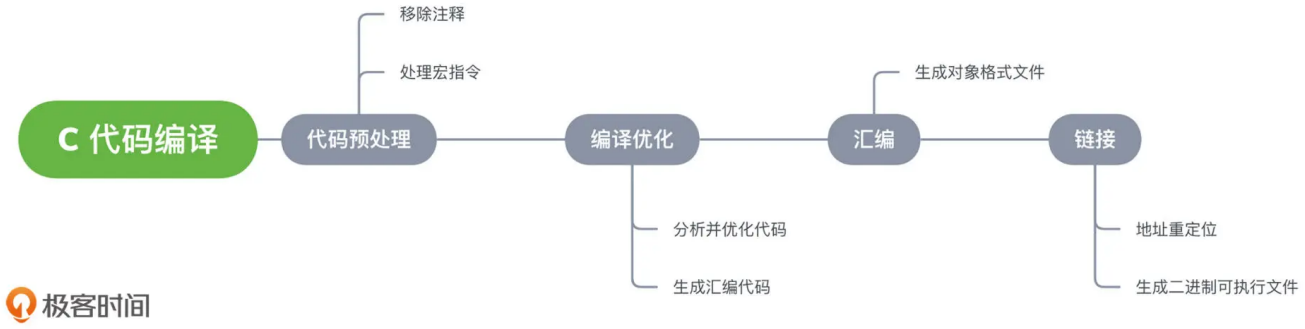
编写完一段 C 代码，接下来的步骤就是对这段代码进行编译了。在执行编译命令时，为了保证程序的健壮性，我们一般会同时附带参数 “-Wall” ，让编译器明确指出程序代码中存在的所有语法使用不恰当的地方。

如果将那段用来回顾核心语法的 C 代码存放在名为 “demo.c” 的文件中，那我们可以使用下面这行命令来编译并运行这个程序：

复制代码

```
gcc demo.c -o demo -Wall && ./demo
```

一般来说，C 代码的完整编译过程可以分为如下四个阶段：



1. **代码预处理**：编译器会首先移除源代码中的所有注释信息，并处理所有宏指令。其中包括进行宏展开、宏替换，以及条件编译等。
2. **编译优化**：编译器会分析和优化源代码，并将其编译成对应的汇编格式代码，这部分代码中含有使用汇编指令描述的原始 C 程序逻辑。
3. **汇编**：编译器会将这些汇编代码编译成具有一定格式，可以被操作系统使用的某种对象文件格式。
4. **链接**：通过链接处理，编译器会将所有程序目前需要的对象文件进行整合、设置好程序中所有调用函数的正确地址，并生成对应的二进制可执行文件。
   编译结束后，我们就得到了可以直接运行的二进制文件。在不同的操作系统上，你可以通过不同的方式来运行这个程序，比如双击或通过命令行。

总结

讲到这里，今天的内容也就基本结束了，最后我来给你总结一下。

这一讲，我们通过一个实例，带你快速回顾了 C 语言的一些重要语法特性。为了方便你复习，我把这些涉及到的核心语法特性总结成了一张表格：

语法（语言）特性	基本功能
入口函数	C 程序的执行入口
数组	用于将同类型数据组织在连续的内存中
结构	用于将不同类型数据组织在连续的内存中
联合	用于在同一块内存中“存放”不同类型的数据
控制结构	提供了程序的逻辑控制能力
指针	用于灵活访问内存
宏	编译前对程序进行预处理
断言	判断某些程序运行所需的假设是否成立
函数内联	用函数体替代函数调用以优化性能
运算符	提供基本计算能力
标准库	提供一系列封装好的函数和系统功能
变量	定义数据
枚举	提供一种可能取值有限的数据类型
类型定义	通过 typedef 关键字定义新类型
静态函数	只在当前编译单元可见的函数

我还带你回顾了一个 C 程序的完整生命周期：代码编写、编译、运行。其中，C 代码的完整编译过程可以分为代码预处理、编译优化、汇编、链接四个阶段。程序的汇编、链接与运行，都会涉及与所在操作系统相关的一系列精细处理过程。我会在这门课的第四个模块中带你深入了解这些内容。

除此之外，我们还从语言本身的角度，探讨了 C 语言与其他编程语言的不同之处。C 语言作为一种命令式编程语言，抽象程度更低，语法结构可以直接由相应的机器指令经过简单的组合来实现。

思考题

最后，我们一起来做一个思考题吧。

在这一讲第一部分的 C 代码实例中，我们为何要给函数 findAddr 添加 `static` 关键字？不添加这个关键字的话，程序是否可以编译运行？欢迎在评论区分享你的思考和见解。

希望这一讲可以让你对 C 语言的基本情况有一个整体的感知。下一讲，我们会从一个 C 程序的基石，数据和量值出发，正式开始我们的 “C 语言核心语法实现”之旅，一起来看编译器是如何在背后实现这些基本语法的。我们下一讲见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 4 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 课前热身 | 学习这门课前，你需要了解哪些知识？

下一篇 02 | 程序基石：数据与量值是如何被组织的？

精选留言 (13)

写留言

ZR2021 神 置顶

2021-12-07

老师，为啥是inline static而不是static inline的？这里有点奇怪

> 作者回复: 这是一个好问题。首先，这两种写法都是可以正常工作的。在 C17 标准之前，declaration specifier 可以按照任意的顺序摆放。而自 C17 标准之后，便规定 storage-class specifier（包含 static）应该被放置在各类定义的最开始，其他形式则为过时的写法。
>
> 6.11.5 Storage-class specifier
> The placement of a storage-class specifier other than at the beginning of the declaration specifiers in a declaration is an obsolescent feature.

6

pedro 置顶
2021-12-09

我尝试去掉了 findAddr 函数中的 static 关键字，如下：
inline /*static*/ const char* findAddr(const CONN* pip)
再次尝试编译，发现完全编译不过：
gcc -o demo -Wall ./example.c
/tmp/ccPazvp1.o: In function `main':…
展开

作者回复: 哈哈哈，实践出真知！

我直接在这里解答一下这个问题。正确情况下是如果在非优化的情况下，inline 需要配合 static 或者 extern 才能够编译。而在优化情况下，inline 可以单独使用。为什么会这样呢？

实际上，在 C17 标准中我们使用的 inline 关键字来源于 C99 标准。而在这个标准中，仅带有 inline 的函数具有 inline definition，表明该函数定义仅用于内联。而在非优化情况下，编译器通常会去寻找对应符号的 external definition（可以通过添加 extern 而获得），因此这种情况下会出现 linker error。而通过添加 static 关键字，可以使得对应函数定义具有 internal linkage，从而被编译器使用。在优化情况下，编译器会直接使用 inline definition 进行内联。

具体可以参考标准：http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf（112页，6.7.4.6）

4

送过快递的码农
2021-12-06

老师，宏和函数的区别是啥？我之前听说他比函数访问更加快？执行上面是否一致呢，函数要进行call，压栈，完事儿出栈，还得ret返回调用前地址 ，但是宏相对简单一点？函数编译需要链接，宏相当于常量，操作系统全局，所以不需要么？函数处理内容更加多，宏比较固定？主要Java程序员没接触过宏，所以感觉是个很陌生的领域
展开

作者回复: C 中的宏你可以简单理解为根据一定规则对源代码进行的字符替换，并且这个过程是在编译器开始处理 C 语法代码之前就会进行的。这部分内容我们会在第 09 讲介绍哈。

共 4 条评论 >

2

qinsi
2021-12-09

C99里inline函数需要两套声明，一个inline的，一个非inline的。例子里的程序如果去掉static的话，如果编译时没有优化就不会用到inline的声明，而如果找不到非inline的声明的话就会报链接错误。

如果不想写两套声明的话，要么加上static（这样就内部链接了），要么加优化参数-O2...

展开

2

勇闯天涯

2021-12-08

老师你好，有什么推荐的C语言书籍吗？偏进阶和高级篇的那种

作者回复: 可以看看《Modren C》？

共 2 条评论

1

ZR2021

2021-12-07

工作5年了，都是纯c开发，看了很多开源代码，内核代码，就是没时间好好总结深入，正好这次趁这门课跟着于老师好好总结学习下，万分期待！！！

展开

作者回复: 哈哈哈，欢迎支持，一起学习！

1

Jerry_Yuan

2021-12-09

const char* findAddr(const CONN* pip)可以过编译，但是同时去掉static和inline变成inline const char* findAddr(const CONN* pip)则不能过编译，但是都可以gcc -S生成汇编代码。后者发生ld error，是不是编译器觉得这个findAddr是从别的地方来的，不是特别安全呢？

展开

作者回复: 可以参考我在另一个问题下的回答哈。

=

2021-12-09

加static是为了让findAddr()函数只在本文件中使用，不能够被其他文件外部调用。在本例中，可以去掉static。

展开

LDxy

2021-12-07

findAddr函数一定要写在main函数的前面吗？

展开

作者回复: 不一定哈，可以在函数调用前先写函数原型，然后函数体就可以放在任意位置了。

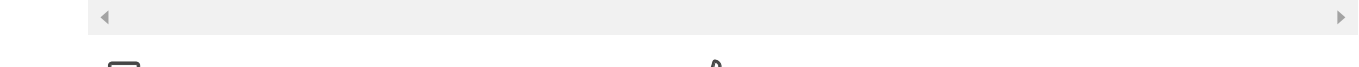
dog_brother

2021-12-07

我们用的还是gcc 4.8.5，用老师介绍的https://godbolt.org/这个工具，很牛啊，原来老师开篇里的c和汇编的对比图是这么来的呀~

展开

作者回复: 正解！



jack123

2021-12-07

应该可以吧，函数加上static，表示仅本文件内访问，不加static默认是extern的，其他文件include的时候，可以被调用到

展开

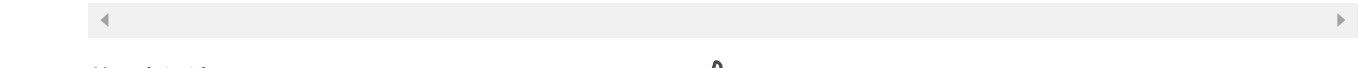
共 2 条评论 >

dog_brother

2021-12-07

老师，命令式编程和声明式编程，是不是可以理解为面向对象和函数式编程啊？

作者回复: 函数式编程是声明式编程的一种，但面向对象属于哪一种类型本身是有争论的。



共 2 条评论 >

dog_brother

2021-12-07

static修饰函数，表示该函数只能被本文件中的函数调用，而不能被同一程序其它文件中的函数调用。在本文的事例里，去掉也是可以的。

展开